(____ cartoon)

5. Pattern Matching and Regular Expressions for Text Processing

"80% of a data analyst's time is spent cleaning up data." `grep()` and `sub()`, below, find lines containing data in text and extract the data from those lines.

Pattern Matching

`grep(pattern, x, ignore.case=FALSE, value=FALSE)` returns a vector of indices of elements of character (string) vector `x` matching `pattern`. `value=TRUE` ⟹ return values instead of indices. ("`grep`" is an acronym for "global regular expression print.") e.g.

```
  a = c("Brown,Joe    123456789 jbrown@wisc.edu      1000",
        "Roukos,Sally 456789123 sroukos@wisc.edu     5000",
        "Chen,Jean    789123456 chen@wisc.edu       24000",
        "Juniper,Jack 345678912 jjuniper@wisc.edu  300000")
  grep(pattern = "j", x = a)
  grep(pattern = "j", x = a, ignore.case=TRUE, value = TRUE)
```

`sub(pattern, replacement, x)` returns a copy of `x` after replacing the first occurrence of `pattern` with `replacement` in each element of `x`. `gsub()` replaces *all* occurrences (`g` indicates *global*). e.g.

```
sub(pattern = "e", replacement = "E", x = a)
gsub(pattern = "e", replacement = "_E_", x = a)
```

Regular Expressions

A *regular expression* describes a set of character strings. In a regular expression,

- letters and digits (`a-z`, `A-Z`, `0-9`) match themselves
- `.` matches any single character
- `\d` matches a *digit* character: `0123456789`

  Note: regular expression escape sequences are written with *one* backslash in R documentation, as in `\d`. But, in an R character string, that one backslash must be typed *twice*, as in `"\\d"`. The first backslash says, "an escape sequence is underway ..." and the second says, "... and the escape sequence is the one for backslash."
- `\w` matches a *word* character: a letter, digit, or `_` (underscore)
- `\s` matches a *space* character: space, tab, and newline (and some others)
- `\D`, `\W` and `\S` negate the previous three classes
- square brackets, `[...]`, enclose a *character class* that matches any one of its characters; except that `[^...]` matches any one character *not* in the class; e.g.

  ```
  gsub(pattern = "[aeiou]", replacement = "", x = a) # strip vowels
  gsub(pattern = "[^aeiou]", replacement = "", x = a) # strip non-vowels
  ```

- `^` matches the beginning of a line (`$` matches the end); e.g.

  ```
  grep(pattern = "^r", x = a)
  grep(pattern = "^r", ignore.case = TRUE,  x = a)
  ```

- `\<` matches the beginning of a word (`\>` matches the end); e.g.

  ```
  grep(pattern = "e\\>", x = a) # note: double backslashes
  ```

- repetition quantifiers in `{...}` indicate matching the previous expression
  - `{n}` exactly `n` times
  - `{n, }` `n` or more times (shorthand: `*` means `{0, }`, `+` means `{1, }`)
  - `{n,m}` `n` to `m` times, inclusive (shorthand: `?` means `{0,1}` or "optional"); e.g.

  ```
  grep(pattern = "\\d{4}$",    x = a) # 4 digits, end-of-line
  grep(pattern = " \\d{4}$",   x = a) # space, 4 digits, end-of-line
  grep(pattern = " \\d{4,5}$", x = a) # space, 4 or 5 digits, end-of-line
  ```

  Note: repetition is maximal, except that appending `?` to a quantifier makes it minimal. e.g.

  ```
  sub(pattern="\\d{1, }" , replacement="X", x=a) # also try "?" after "}"
  ```

- parentheses, `(...)`, enclose an expression; a *backreference* `\N` (where `N` is in `1:9`) refers to what the `N`th enclosed expression matched; e.g.

  ```
  link = "blah blah blah ... <a href=http://www.google.com>Google</a> blah ..."
  sub(pattern=".*<a href=(.*)>.*"   , replacement="\\1", x=link) # match too much
  sub(pattern=".*<a href=(.*?)>.*"  , replacement="\\1", x=link) # one fix
  sub(pattern=".*<a href=([^>]*)>.*", replacement="\\1", x=link) # another fix
  # rewrite "last,first ID email ..." to ".csv": "first,last,user,ID"
  b = sub(pattern = "(\\w+),(\\w+) +(\\d+) (\\w+).*", replacement = "\\2,\\1,\\4,\\3", x=a)
  ```

- `|` means *or*; e.g.

  ```
  grep(pattern = "Joe|Jack", x = a)
  grep(pattern = "J(o|a)", x = a)
  ```

- `. \ | ( ) [ { ^ $ * + ?` are *metacharacters* with special meaning; to use them as regular characters, *escape* them with `\` (doubled, as described above)

`?regex` has more information.

Splitting Strings

`strsplit(x, split)` splits each string in character vector `x` on regular expression `split`. e.g.

```
strsplit(x=a, split=",")
strsplit(x=a, split=" +")
strsplit(x=a, split=",|( +)")
```